LAB 3

CODING TIPS AND OTHER ADVICE

The most important preparation you should do for Lab 3 is carefully read the lab manual. Browse through it on your first pass. Read it and pay attention to more details on the second pass. Then read the header files and reference sheets it refers to. Take notes about some of the programming details and suggestions.

The tips and advice in this document are intended to reinforce and/or supplement the lab manual and other resources based on observations and feedback from past semesters.

Using the cyBot_getByte() function

This function is in cyBot_uart.h. It is a blocking function, which means that it "blocks" or "waits" until it receives something. There is a busy-wait loop inside the function implementation (not visible to you as a precompiled library) that repeatedly checks if a byte (character) has been received. Thus the function will not return until a user presses a key in PuTTY. The function then returns an integer representing the ASCII code of the key pressed. Keep this in mind as you are running your program – if it doesn't appear to be doing anything, maybe you need to press a key.

Writing a send string function

You will be writing messages to the PC in this and future labs, thus it will be very handy to have a function for this. The cyBot_sendByte() function in cyBot_uart.h sends only one character. It does not send a string of characters. Write your own function to send a string. You will also want to use the C standard library function sprintf() to format variables into a string. This is especially important when you are sending integer or floating point values for the sensor data.

Using the cyBOT_Scan_t struct

In the Lab 3 code, there is a struct for PING and IR data readings.

The type definition for the struct is in cyBot_Scan.h: cyBOT_Scan_t

You will need to create a variable of this type for the scan data. Note that in Lab 2, the oi_alloc() function created the sensor data struct and returned a pointer to it. Then you used the pointer to the struct and arrow notation to reference struct members (`struct pointer->struct member`). If you look at the oi_alloc() function code in open_interface.c, you will see it uses the calloc() function (from the C standard library). You could also use calloc() in your own code. If

so, then your struct code and parameter passing would look similar to Lab 2. And don't forget to use the free() function (from the C standard library) before exiting your program!

Alternatively, you could declare a struct variable like any other variable. If so, you are using a struct variable, not a pointer to a struct. Keep in mind: a) the struct variable name is not a pointer, b) passing a struct variable as a call by reference parameter to a function must use the & (address) operator, and c) referencing a struct member uses dot notation (`struct variable name.struct member`).

For example, notice the struct parameter here:
void cyBOT_Scan (int angle, cyBOT_Scan_t* getScan);

The * means that getScan is a pointer to cyBOT_Scan_t struct. In other words, it is an address. That's because it is a call by reference parameter which allows the struct members to be updated by the function. So when you call the function, the second parameter needs to be an address to the struct. Such as the following sample line of code to call the function:
cyBOT_Scan (currentAngle, &currentScan);

Using the PING sensor data

We recommend starting with the PING sensor data only in Lab 3 to simplify your data analysis. The IR sensor data will be important for accurate navigation, but more background information is required to interpret that IR data returned by the scan function. It's okay if students use the IR sensor, but TAs will first focus on helping students with the main functionality using the PING sensor.

The PING sensor does a good job of estimating distance to the object. However, there are many limitations of the PING sensor (or any real device we might use). The PING sensor will not precisely measure where the edges of an object are and the width of the object, in absolute terms. However, when comparing measurements from the PING sensor for different objects, the data should correctly show which objects are narrower or wider.

Thus, the PING sensor should be sufficient to get rough estimates of objects in front of the robot for this lab. It isn't the best choice alone for some of the measurements. However, it is good enough for the purposes of this lab. Keeping in mind the cone-shaped sound wave that the sensor emits (see the sensor datasheet), you will need to have objects spaced far enough apart when placed in front of the robot in order to detect them as different objects. Also, the maximum range for the PING sensor as given in its datasheet is about 3 meters. Keep this in mind when analyzing the data. Ignore data that would be outside of this range or the cone shape as either invalid data or noisy data. You can test this by having no objects within, say, 2.5 meters.

Real devices have operating specifications and limitations in real environments. Dealing with limitations helps us learn about them. For this lab, we decided that a good learning outcome would be seeing some of the limitations of a PING sensor – see what real sensor data look like, see pros/cons of a particular sensor, consider when other solutions are needed. If the sensor data

don’t distinguish between a narrow and wide pillar object, then try spacing them farther apart, or making a wider object by grouping pillars together. If the PING sensor still doesn’t provide different data, then it’s okay to demonstrate that.

The goal for this lab includes getting familiar with using a sensor, recognizing what it can and cannot do, and trying to accomplish some tasks with the sensor given its noisy, imprecise, imperfect data. This work should also provide some motivation for wanting better data, needing a better sensor for some measurements, and to some extent, thinking about how to "clean up" sensor data in software and/or hardware. You are not expected to get better data in this lab, but now you have some awareness for future labs and the project – you won't be surprised by the data in the future.

Seeing the forest and the trees

Have you heard the expression, “Can’t see the forest for the trees.” We don’t want to get too involved in the details of a problem and miss the bigger picture. The challenge for us is that we need to pay attention to many details to get things to work in the lab. You need to follow the PuTTY setup and servo calibration procedures very carefully to send a message to the PC or determine where an object is located. You need to use the correct syntax with a struct. You need use functions correctly. And so on.

At the same time, we want you to step back and think about the system. Something may not work perfectly. There may not be a “correct” answer or “correct” data. This lab is an opportunity to get more comfortable with this. Engineering is about understanding the system we are working with, and iterating and optimizing to find solutions that will perform according to requirements. We may not find that solution right now, but we are better prepared to later.